二叉堆

2019年3月8日 16:37

-
- 二叉堆

1. 二叉堆：一种支持插入、删除、查询最值的数据结构
   1) 是满足“堆性质”的完全二叉树
   2) 小根堆性质：父节点的权值小于所有子节点权值
   3) 大根堆性质：父节点的权值大于所有子节点权值
2. 数组法存储完全二叉树：
   1) 下标0留空，以1作为根节点
   2) 根节点下标*2为左子，下标*2+1为右子
   3) 下标/2（整除）为父节点下标（>0时）
   4) int heap[SIZE],n=0;//堆数组和数据量
3. void up(int p){ //将第p个节点上移到合适的位置
   while(p>1){
   if(heap[p]>heap[p/2]) //不满足大根堆性质时
   swap(heap[p],heap[p/2]);
   else
   break; //已经合适了
   p/=2;}
   /*存储效率稍高的版本
   int t=heap[p];
   while(p>1 && t>heap[p/2]){
   heap[p]=heap[p/2];
   p/=2;}
   heap[p]=t;
   */}
4. void insert(int val){ //插入值为val的数据
   heap[++n]=val; //先插在最后，并增加计数
   up(n); //上浮
   }
5. int getTop(){ //返回根节点数值
   return heap[1];
   }
6. void down(int p){ //将第p个节点下移到合适的位置
   int s=p*2; //存储较大孩子的下标
   while(s<=n){ //左子是存在的
   if(s<n && heap[s+1]>heap[s])
   ++s; //右子是存在的，而且更大

```
        if(heap[p]<heap[s]) //不满足大根堆性质时
            swap(heap[p],heap[s])
        else
            break; //已经合适了
        p=s;
        s=p*2;}
}
```

7. ```
   void extract(){          //移除顶堆
       heap[1]=heap[n--]; //删除顶堆同时减少计数
       down(1);       //调整新顶堆（旧末尾）
   }
   ```
8. ```
   void remove(int p){  //移除第p个数据
       heap[p]=heap[n--];
       up(p),down(p);         //不能确定上还是下，都做一遍
   }
   ```
9. STL的优先队列**大根堆**
   1) template<class T,class Container = std::vector<T>,class Compare = std::**less**<typename Container::value_type> >class priority_queue
   2) 即**priority_queue**<类T,容器=std::vector<T>,比较函数=std::less<T> >
   3) push(x)压入x；top()返回顶；pop()删除顶
   4) 类型T的operator<(l,r)实现为l>r时会变成小根堆
      -
      - 例题

1. 贪心：k张优惠券可让n个货品价格p变成c，求身带m元可买几个货（USSTD3D）
   1) 引：p最小和c最小各用小根堆qpqc维护（第一维是价格，第二维是下标）；k张优惠券能节约的价格也用一个小根堆qd维护（优惠券下标就不用存了……）
   2) 引：因为钱是固定的，c是不大于p的，所以每次先贪心比较c最优惠的有没有必要用优惠券买，通过qd的最小值可得到用得最亏的优惠券花了多少，考虑要不要把“反悔”，把那个优惠“放弃掉”，用在这个c上。未决定怎么用优惠券的初状态，只需给qd插入初值k个0即可
   3) 转：如果反悔不起，就转去考虑不优惠时最便宜的p
   4) 结：用布尔数组判断各物品买了与否，当买不起最便宜的物品或买完时输出
   5) ```
      int n,k;
      ll m;    //money
      int p[MN],c[MN],ed[MN];        //原价，优惠价，决策结束与否
      typedef pair<int,int> pii;          //权值，物品编号
      priority_queue<pii,vector<pii>,greater<pii> >qp,qc;
      //qp关于原价p的小根堆，qc关于优惠价c的小根堆，qd关于折扣=原价-优惠价的小根堆
      priority_queue<int,vector<int>,greater<int> >qd;
      ```
   6) ```
      for__(i,1,n)
          scanf("%d%d",p+i,c+i),
      ```

```
        qp.emplace(p[i],i),
        qc.emplace(c[i],i);
    for_(i,0,k)         //k张优惠券带来的折扣，初值是没开始用优惠券（=0）
        qd.push(0);
    for_(i,0,n){
        while(qp.size() && ed[qp.top().second])
            qp.pop();          //清掉已确认项，确保首项未确认
        while(qc.size() && ed[qc.top().second])
            qc.pop();          //清掉已确认项，确保首项未确认
    //      if(qc.empty() || qp.empty())      return !puts("出锅啦");
        if(qd.top()+qc.top().first < qp.top().first){          //这张优惠券的用法亏了
            int t= qc.top().second;              //决定买优惠后最便宜的编号t的物品
            int cst= c[t]+qd.top();   //优惠券改买t，要多花的钱
            if(m>=cst)         //cout<<"\n优惠买："<<t,        //买得起
                qd.pop(),          //这张优惠券过去的用法作废
                qd.push((p[t]-c[t])),      //优惠券新节约了这么多钱
                ed[t]=1,//决定用优惠券买它
                m-=cst;            //买买买！
            else{    //cout<<m<<"优惠也买不起"<<cst<<"的"<<t;
                return !printf("%d",i);}  //买不动啦
        }else{              //优惠券用得不亏，只能原价买
            int t= qp.top().second;              //决定买优惠前最便宜的编号t的物品
            int cst= p[t];     //直接买t，要多花的钱
            if(m>=cst)         //cout<<"\n直接买："<<t,        //买得起
                ed[t]=1,//决定直接买它
                m-=cst;            //买买买！
            else{    //cout<<m<<"直接买不起"<<cst<<"的"<<t;
                return !printf("%d",i);}  //买不动啦
        }                   //cout<<"  第"<<i<<"轮还剩"<<m<<"钱\n";
    }printf("%d",n);            //土豪……全买得动
```

2. 贪心：从一堆数中任取出两个，做乘法，再加一，再塞回，求最后剩下的一个数最大，最小可能是多少
   1) 求最大：每次取出最小的两个，再塞回，即可
   2) 求最小：每次取出最大的两个，在塞回，即可
   3)
```
int n, t;
priority_queue<ull>q;
priority_queue< ull, vector<ull>, greater<ull> > q2;
scanf("%d",&n);
for_(i,0,n){
    scanf("%d",&t);
    q.push(t);
    q2.push(t);}
while(q.size()>1){
    ull t1= q.top();   q.pop();
    ull t2= q.top();   q.pop();
    q.push(t1*t2+1);}
```

```
while(q2.size()>1){
	ull t1= q2.top(); q2.pop();
	ull t2= q2.top(); q2.pop();
	q2.push(t1*t2+1);}
cout<<q2.top()<<" "<<q.top();
```

3. 两长度为N的序列中任各取一数的最小的N个和
   1) 可以让i，j分别指向序列a和b的某数的下标
   2) 输出a[i]+b[j]后，新出现的可能解是a[i+1]+b[j]或a[i]+b[j+1]
   3) i和j交替增加时可能导致<i+1,j+1>重复出现，为解决这个问题，可将ij二元组改为ijb三元组，可以随意加入的是<0,0,b>的b初值不同的值，当且仅当取出的b与初值b相同时，可推入与初值b相同的<i,j,b>
   4)
```
int a[100005],b[100005];
class S{
	public:
		int i,j;
		bool changed_i;//这次偏移的是i
		bool operator<(const S s)const{
			return a[i]+b[j]>a[s.i]+b[s.j];
		}
		S(int i,int j,bool b):i(i),j(j),changed_i(b){}
};
priority_queue<S>q;

int main() {
	int n;
	scanf("%d",&n);
	for_(i,0,n)
		scanf("%d",a+i);
	for_(i,0,n)
		scanf("%d",b+i);
	sort(a,a+n);
	sort(b,b+n);
	q.push(S(0,0,true));
	while(--n){
		S s=q.top();
		q.pop();
		printf("%d ",a[s.i]+b[s.j]);
		q.push(S(s.i,s.j+1,false));
		if(s.changed_i)
			q.push(S(s.i+1,s.j,true));}
	S s=q.top();
	printf("%d",a[s.i]+b[s.j]);
	return 0;}
```
   5) 还有一个好理解但很慢的方法
   6)
```
map<pair<int,int>,bool>exist;
for_(i,1,n){
	S s=q.top();
	q.pop();
	printf("%d ",a[s.i]+b[s.j]);
	S s2(s.i+1,s.j);
	S s3(s.i,s.j+1);
	if(!exist[make_pair(s2.i,s2.j)])
		q.push(s2);
	if(!exist[make_pair(s3.i,s3.j)])
```

```
            q.push(s3);
        exist[make_pair(s2.i,s2.j)]=1;
        exist[make_pair(s3.i,s3.j)]=1;}
```

i.

ii.

iii.

iv.

v.

vi.

vii.

viii.

ix. --------我是底线----------